060615/Thomas Munther
IDE-sektionen
Högskolan i Halmstad

Laboratory 6

This laboratory will deal with analysis of functions, interpolation, curve fitting, integrals and differential equations.

We will need to use polynomials and therefore we have to be familiar with the representation of these. A general polynomial looks like:
$p(x)=a_n x^n + a_{n-1}x^{n-1} + \ldots\ldots\ldots + a_1 x + a_0$
and is represented by a vector in matlab:
$p=[\ a_n\ \ a_{n-1}\ \ldots\ldots\ a_1\ \ a_0\ ]$

Below we have smaller list of commands dealing with polynomials.

polyval(p,x)	Calculates the value of polynomial p for different x. If x is a vector then the polynomial is evaluated for each element in the vector x.
poly(A)	Gives a vector that represents the characteristic polynomial for the matrix A.
Roots(p)	Gives a vector with the zeros for the polynomial p(x)=0.
polyder(p)	Gives a vector that represents the time-derivative of the polynomial p(x). The coefficients are sored in the vector p.
conv(p,q)	Multiplies the polynomials p and q with each other. Returns a coefficient vector.
polyint(p)	Integrates the polynomial p analytically and uses the constant of the integration c. The constant c is assigned to 0, if it is not explicitly given.
residue(p,q)	Makes a partial fraction expansion of p(x)/q(x).

Example:
$p(x)=3x^3 + 2x^2 -2x + 4$

Represent the polynomial in Matlab and find its zeros.
Plot the function and check the zeros!
This gives a quick idea of what the function looks like. See Figure 1!

```
>> x=-10:0.1:10;
>> plot(x,3*x.^3+2*x.^2-2*x+4), title('p(x)=3*x^3+2*x^2-2*x+4')
```

>> xlabel('x'), grid
Define the polynomial. The coefficients in the polynomial are arranged in descending order in the vector p. The orders that are nonzero in the polynomial will be represented by zeros in the vector p.

>> p=[3 2 -2 4] % Represents the coefficients in p(x)
With *polyval* we can easily calculate the value of the polynomial in different x-values.

>> polyval(p,6), polyval(p,7), polyval(p, -5) % polyval(p,[6 7 -5]) also works !

ans= 712 , ans = 1117 , ans = -311

Are these values correct, if we use the plot below?
Make some readings and check your result!

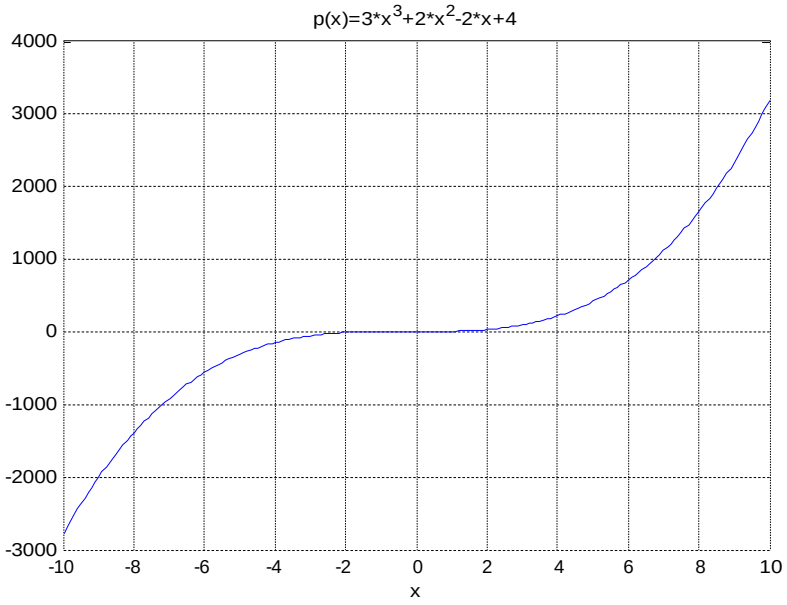

Figure 1

Let us try some of the other functions that can be applied to polynomials, like *polyder and polyint.*.They perform the time-derivative and integrate the polynomials

$p(x)= 3x^3 + 2x^2 -2x + 4.$

The time-derivative becomes:

$p'(x)= 9x^2 + 4x -2$

and integration gives:

$P(x)= (3/4)x^4 + (2/3)x^3 - x^2 + 4x+C$

Now compare what Matlab gives us :

```
>>  C=1          % C is a integration constant.
>> Pder=polyder(p), Pint=polyint(p,C)

Pder =
             9     4    -2
Pint =
```

0.7500 0.6667 -1.0000 4.0000 1.0000

Note that we only obtain the coefficients in the new polynomials !

Introduce another polynomial q(x)=x .

>> q=[1 0]

Multiply the polynomial q(x) with p(x), it becomes:

$pq(x)= 3x^4 + 2x^3 - 2x^2 + 4x$

Matlab gives:

>> conv(p,q) % Performs a polynomial multiplication of p and q.

ans= 3 2 -2 4 0

Let us continue with other functions. Now, check the zeros in the polynomials.
This is done with the Matlab command *root.*

roots(p) % Gives a vector with zeros to the polynomial p(x).

ans =

-1.6022
0.4678 + 0.7832i
0.4678 - 0.7832i

Above we can see something quite obvious. There are 3 zeros to a third order polynomial. It is nothing to be astounded by, but only one of these zeros is real.

Can we foretell this by looking at the plot in Figure 1?

Yes, I would say so, because if we zoom the curve, we can find the zero-crossing. This gives us a real-valued zero. In our example there is only one, but what happens to the other two zeros? Since they are complex-conjugated, they are not visible.

Finally we will also take a look at the *residue* command. It makes a partial fraction expansion of the polynomials p(x) and q(x).
Look at the ratio q(x)/p(x)!

```
 [t,n, the_rest]=residue(q,p)  % There are 3 output arguments from  residue.
t =
                -0.1090
                 0.0545 - 0.0687i
                0.0545 + 0.0687i

n =
                -1.6022
                0.4678 + 0.7832i
                 0.4678 - 0.7832i
the_rest =
                []
```

A partial fraction expansion looks like:

$R(x)= t_1/ (x-n_1) + t_2/ (x-n_2) + t_3/ (x-n_3) + the\_rest$

Now let us define a function in Matlab. As you hopefully remember this is nothing more than a m-file. We will call it *func.m* and it should be used together with an input argument x.
X is a real-valued argument.

```
-------------------------------------
% The function func.m created 050621, Thomas Munther
function  f=func(x)

f=3*sin(x)-2+cos(x.^2);
-------------------------------------
```

We will now take a look at a plot of the function, but first we must decide what region we are interested in. See Figure 2 !

```
>> x=-10:0.1:10;
>> plot(x,func(x)), grid on
>> title( 'func(x)')
```

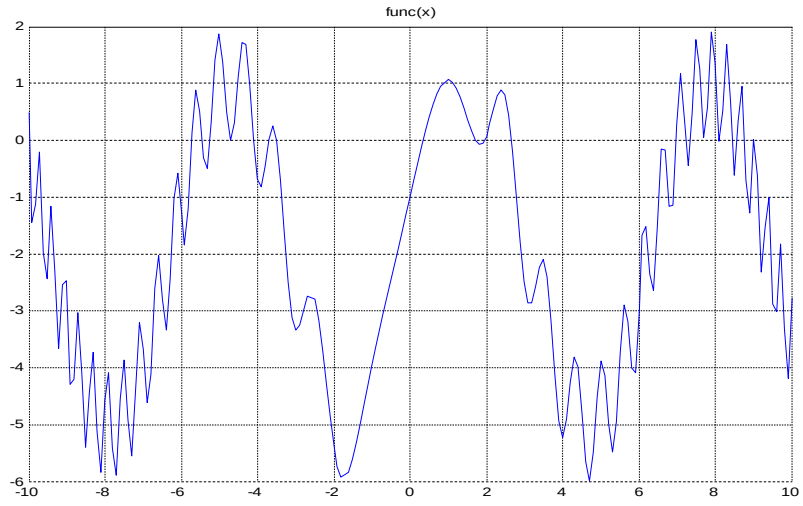

Figure 2

In the previous example the zeros for a polynomial were decided with the command roots.
Here on the other hand we have no polynomial, but only a function, and we can instead use the command *fzero*. The *fzero* command uses a repetetive algorithm. We must always add an initial guess and then the *fzero* tries to localize a zero closest to the guess.
Assume we would like to find the zero in the interval 0-1 in the example above.
The function has an infinite number of zeros outside this interval 0-1.

Our guess becomes :

>> fzero(@funk, 0.5), fzero(@funk, 0.9), fzero(@funk, -1.5)
They all produce the same zero!

ans= 0.3423

Our three guesses seems to use the fact that all of them have the same sign of the time-derivative, which and is why the algorithm converges toward 0.3423.
If we change the initial guess to be on the other side of the peak, fzero will give us a new answer (zero).
>> fzero(@funk, 1.5)

ans = 1.7053

Localize minima and maxima of functions
Let us try to find the local minima and maxima for the function func(x).
The interval of interest is [-6 0]. The algorithms are iterative. There are 2 methods to use. The first one decides x in a given interval, and the second one looks for x around an initial guess.
To decide the maxima we are looking for an x that minimizes the negative function: -func(x).

fminbnd(func,x1,x2)	Gives the x-value for a local minima.
[x,y,flag,info]=fminbnd(func,x1,x2)	As above, but we also get the y-value. Flag gives a positive number, if minima is localized and a negative number otherwise. We store the information about the iterations in the variable *info*.
fminsearch(func, x0,)	Gives the x-value for a local minima somewhere close to the initial guess x0.

Decide the global minima and maxima that exist on the interval -8 to 0 for the function func(x).
This gives:
x =
-1.7326

y =
-5.9511

flag =
1
This seems to be true for our function.
If we want to find the maxima values, the same command can be used.
The only difference will be a minus sign in front of the function.

Whenever we call for the function fzero. The command looks for a minima but will in fact localize the maxima due to the minus sign. See below !

```
function  f=func(x)
    f=-(3*sin(x)-2+cos(x.^2));          % Note the minus sign in front of the parenthesis.
>> fminbnd(@func,-8,0)
```

Look below for the answer. Note that the y-value is negated. Compare the plot func(x) to our result below.

```
x =                         y =
  -1.8691                     -5.0046
```

Our maxima occurs at x=-1.8691, and the value of y becomes y=5.0046.
Is this an accurate result according to your opinion?

Interpolation of data sets

Assume that we have made a number of measurements and thereby guessed a function relating the input and output. Then we also have some knowledge of the values between the measured points. This is very common in sampled applications. Think of a sampled continuous sine wave function with a very low sampling frequency.
This can be done very simply, if we use a vector x with low resolution. Let us create one. Use the x vector values and calculate the corresponding sine function values and plot it in Figure 3.

```
>> x=0:20;
>> y=sin(x), plot(x,y),grid
```

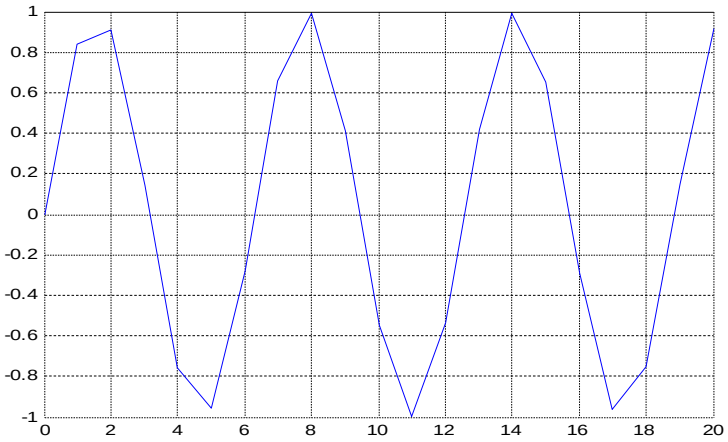

Figure 3

We could think of it as a rather badly shaped sine wave curve probably due to a too long sampling period.
By interpolation one decides on a function P(x) that passes through certain known data points. The interpolated function can be used to calculate the function value approximately for any x. Normally one uses polynomials or functions combined from polynomials (spline functions) to make interpolations, but also sine and cosine waves can be useful.

Now, let us see what happens when we try to make a curve fit 10 random numbers.

```
>> y=rand(1,10)          %   10 random numbers in a row vector
```

```
>> plot(y,'*')            %
```

Investigate whether it is possible to find a polynomial that can be adjusted to the random data.
Enter the figure window under Tools-> Basic Fitting.
Mark the ninth-degree polynomial !
This gives a polynomial that passes through all random data points.
The basic fitting uses the least square method. Even if we use a lower degree it is still the best solution according to the least square method.
Mark the small box *show equation*. We can then explicitly see the polynomial in the figure window.
Remove the ninth-degree polynomial and choose a linear equation instead !
Matlab will now try to find the best linear equation.
Mark the small box *plot residuals* !
The error residual between each data point and the linear equation is calculated and shown. Despite large discrepancies this is the best first-order polynomial.
Close the figure window.

We will now try to accomplish the same thing with Matlab commands that we did with the interactive basic fitting in the figure window.

```
y=rand(1,10); x=1:10;

% polynomial fits (least square method, remember last excercise)
p1=polyfit(x,y,1);
p5=polyfit(x,y,5);
p9=polyfit(x,y,9);

% Interpolated data by piecewise (local) polynomials
pL = interp1(x,y,'linear','pp');
pS = interp1(x,y,'spline','pp');

xx=1:0.01:10;  % Improves the resolution in the x-vector.

p1_func=polyval(p1,xx);  % Samples the polynomial
p5_func=polyval(p5,xx);
p9_func=polyval(p9,xx);

pS_func=ppval(pS,xx); % The interpolated functions are piecewise
pL_func=ppval(pL,xx); % polynomials : require ppval, instead of polyval

subplot(2,1,1);
plot(xx,p1_func); hold on
plot(xx,p5_func,'-.');
plot(xx,p9_func,'--');
plot(y,'*');
title('Fitting by Polynomials');
legend('linear','5th degree','9th degree');

subplot(2,1,2);
plot(xx,pL_func,'r'); hold on;
plot(xx,pS_func,'r-.');
plot(y,'*');
title('Interpolation by Polynomials');
legend('linear','cubic spline')
```

In the code above, we make use of the Matlab function “interp1”. It is a specific function for performing interpolations of data. We use polynomials also here, but they are now piecewise polynomials, meaning there are many small local functions describing the transition from one data point to the next (you can think of this as roughly one function fitting per location). This is interpolation using polynomials.

Whether one does a global fitting of a curve, or use local functions for interpolation, depends on the problem one tries to solve. For example, if one wants to change the size of the data (as when making an image bigger in photoshop) one would like to use local piecewise polynomials. If one wants to use the data to predict future events, or to say something about linear or higher relationships, then one would do the global curve fitting. An example of the global curve fitting would be to predict how an electrical device will behave under rising temperatures. One would perhaps not want to expose it to extremely high temperatures, but reasonable low ones and measure the quality of the device. The globally fitted function can then be evaluated beyond the

range of the original samples.

Integrals

Matlab can solve integrals and differential equations with numerical methods.
Numerical integration in Matlab is performed by the command *quad*. This stands for numerical quadrature.
We can solve integrals only with defined limits. This can be done for single, double and triple integrals.
To solve generalized integrals or integrals expressed with symbols, we must use the Symbolic Math toolbox(will come later in the course).

Let us exemplify with some numerical examples:

Example 1:
Decide the integral

$$\int_0^{10} \sqrt{e^x} dx$$

First we put the integrand in a function file: *integrand.m* .

```
-------------------------------------------------
        function y=integrand(x)
        y=sqrt(exp(x));
-------------------------------------------------
```

Write !

```
>>    I1=quad('integrand',0,10)
```

Matlab gives:

```
>>    I1 = 294.8263
```

It seems reasonable enough, if we plot the function.
We can also use *quad* with a fourth argument, namely the tolerance for the calculation. This decides when the numerical integration will stop.

Example 2:

Decide the double integral

$$\int_0^1 \int_0^1 \sqrt{e^x}\sqrt{e^y}\,dxdy$$

Make as in the previous example a function where we put the integrand. We name the m-file *integrand2.m*

```
function z=integrand2(x,y)
z=sqrt(exp(x)).*sqrt(exp(y));
```

Let us now call the function *dblquad* instead !

```
>> I2=dblquad('integrand2',0,1,0,1)
```

and the answer is:

```
I2=1.6834
```

We will now take a look of the integrated area.

```
>> [X,Y]=meshgrid(0:0.1:1,0:0.1:1); % Makes a grid of calculation points.
>> Z=integrand2(X,Y);               % Calculates the integral.
>> mesh(X,Y,Z) ;                   % Makes a plot that connects these points.
```

Matlab can also perform triple integrals using *triplequad* !

Tasks

Solutions that do not fill the following requirements ***EXACTLY, PRECISELY AND TO THE LETTER*** will not be considered:

- Send your solutions by email to:

stefan.karlsson@hh.se

subject: Matlab, Exercise X, YourNames

- Send all the files that are requested, no more and no less, in one single zip file per exercise, with NO sub-folders in the zip file. All the files, for all the tasks should be bundled into one zip file.

- Put the Names of the authors, in remarks, at the top of every m-file.

- Send the solutions within 2 weeks of every exercise session. That is, you have a two week deadline to hand it in.

- You will get 2 chances to send it in to me correctly.

1. In the table below we have some resistance measurements made on an electronic component of resistance. The measurements has been done at different temperatures T.

T ($^{\circ}$C)	R(Ohm)
20.5	760
33	820
51	871
72	940
96	1038

   Make a curve fit to the table data.

   Make a function **e6_1.m** that takes as argument a new value for temperature, and uses the fitted function to estimate the new resistance.

2. Calculate the area confined by the functions
   f(x)=x+3
   and
   g(x)=tan(x)

   in the first positive interval. From x=0 to the first positive intersection point (where f(x)= g(x)).
   Write an m-file **e6_2.m** that outputs the area. Calculate all parts of your solution with Matlab functions, calculate by hand only if you want to verify your answer.

3. Decide all the zeros to the function :

   $0.1x^3 - 5x^2 - x + e^{-x} = -4, \quad x \geq 0$

   It is permitted to use interactivity in order to get approximate values. This could be either keyboard input or a mouse button click in a figure window(although you should use Matlab built in functions to get precision to any initial guesses).
   You should also try to find the minima on the interval $x \geq 0$.
   Function **e6_3.m** should output a figure that displays the function together with the zero crossings as well as the minima postion. Use "legend" on the figure.

4. Create an m-file that reads 2 polynomials (not higher than order 5) and afterwards plots these.
   Find the zeros to the polynomials and take the time-derivatives and also makes the integration.
   Use subplot with 2 windows. One window for each polynomial together with the time-derivative and its primitive function.